模块三　MCS－51 单片机指令系统

【教学聚焦】

知识目标：

1、熟悉单片机系列指令；

2、掌握汇编语言及其格式；

3、掌握基本的寻址方式；

4、掌握基本指令及其功能。

技能目标：

1、 能够熟练的编写调试实验程序；

2、 能够利用单片机仿真器开发调试单片机应用程序的过程；

3、 能够掌握单片机编程器的使用方法；

4、 能独立进行一般性的程序设计。

教学重点：单片机应用的系列指令；

教学难点：能自己编写调试实验程序。

【课时建议】8 课时

项目 3．1 指令系统的基本概念

3.3.1 机器代码、指令系统

机器代码是指计算机能够识别的二进制代码 0 和 1。由于构成计算机的电子器件特性所决定，计算机只能识别二进制代码。

单片机和所有微型计算机一样，要执行某周操作或运算时，要先向 CPU 输入以二进制数为代码的操作命令，这种操作命令就称为指令。指令也是组成程序的基本元素。不同类型的 CPU 都有一套适用于它本身的指令集合，我们把这种指令集合称为该 CPU 的指令系统。同样，MCS-51 系列单片机也有一套指令集和，即 MSC-51 指令系统。MCS-51 指令系统，不但适合于 Intel 公司生产的 MCS-51 系列单片机，而且也适用于其他公司生产的 8051 系列单片机。

MCS-51 指令系统共有各种指令 111 条。其特点如下：

（1）指令执行时间快；

（2）指令短，约有一半的指令为单字节指令；

（3）用一条指令即可实现 2 个一字节的相乘或相除；

（4）具有丰富的位操作指令；

（5）可直接用传送指令实现端口的输入输出操作。

按指令机器码的长度，MCS-51 单片机指令可分为单字节指令、双字节指令、三字节指令。指令的字节越少，所占用的程序存储器空间也越少，所以编程时应尽可能选用字节少的指令。 单字节指令的格式由 8 位二进制编码表示；双字节指令的格式由两个字节组成，操作码和操作数。三字节指令的格式中第一个字节为操作码，后两个字节为操作数。

3.3.2 程序设计、机器语言

程序设计是指设计、编制、调试程序的方法和过程。程序设计往往以某种程序设计语言为工具，给出这种语言下的程序。程序设计给出解决特定问题程序的过程，是软件构造活动中的重要组成部分

软件程序（通常简称程序）是指一组指示计算机每一步动作的指令，通常用某种程序设计语言编写，运行于某种目标体系结构上。打个比方，一个程序就像一个用汉语（程序设计语言）写下的红烧肉菜谱（程序），用于指导懂汉语的人（体系结构）来做这个菜。 通常，计算机程序要经过编译和链接而成为一种人们不易理解而计算机理解的格式，然后运行。

机器语言是指直接用机器码编写程序，能够为计算机直接执行的机器级语言。机器码是一串由二进制代码“0”和“1”组成的二进制数据，执行速度快。但对于使用者来说，用机器语言编写程序非常繁琐，不易看懂和记忆，容易出错。机器语言一般只在简单的开发装置中使用。

3.3.3 汇编语言及指令格式、常用符号

1. 汇编语言

汇编语言是指用指令助记符代替机器码的编程语言。程序结构简单，执行速度快，易优化，编译后占用的存储空间小，能充分发挥单片机的硬件功能，是单片机应用系统开发中最常用的程序设计语言。对于复杂的应用来讲，使用汇编语言编程复杂，程序的可读性和可移植性不强。只有熟悉单片机的指令系统，并具有一定的程序设计经验的用户才能编写出功能复杂的应用程序。对于实时测控系统的单片机来说，采用汇编语言编程最为方便。

指令是指挥计算机工作的命令，是计算机软件的基本组成单元。指令有机器指令和汇编语言两种。机器指令是用二进制数表示的能直接能被计算机识别并执行的指令，由于二进制书写起来较长，通常用十六进制数表示。显然这种指令不便于记忆和理解，书写起来也容易出错。为了便于记忆和使用，常以指令的英文名称或缩写形式作为助记符来表示指令的功能，这样的指令称为汇编语言指令。

2. 指令格式

在 MCS-51 指令中，一条指令主要由操作码、操作数组成。指令的格式如下：

［标号：］ 操作码 ［目的操作数］ ［，源操作数］ ［；注释］

例如， LOOP : ADD A , #50H ；执行加法

在指令格式中，方括号中的内容为可选项，不一定都有。每个区段之间要用分隔符分开：标号与操作码之间用“：”隔开，操作码与操作数之间用空格隔开，操作数与注释之间用“；”隔开，如果操作数有两个以上，则在操作数之间要用逗号“,”隔开 （乘法指令和除法指令除外 ）。

（1）标号表示该指令所在的地址。并不是每条指令都必须有标号，通常在程序分支、转移等需要的地方才加上一个标号。标号是以字母开始的，由 1～8 个字符（字母或数字）组成，不能使用汇编语言中已经定义过的符号名，如指令助记符、寄存器名、伪指令等。标号以“：”结尾。特别注意的是，在一个程序中不允许重复定义符号，即同一程序不能在两处及两处以上使用同一标号。

（2）操作码在前，规定指令所完成的功能，指明执行什么性质和类型的操作。例如：数的传送、加法、减法。

（3）操作数规定操作的对象，操作数可以是一个具体的数（立即数），也可以是这个数据所在地址。

例如：MOV 40H , #30H **75H 40H 30H** ；把数 30H 送给片内 RAM 的 40H 单元内。

前是用助记符表示的指令，后是其对应的机器码。MOV 是操作码，40H 是第一操作数（目的操作数），#30H 是第二操作数（源操作数）

（4）注释字段可有可无，是用户为阅读程序方便而加的解释说明。注释以“；”开始，不影响程序的执行。

3. 指令中常用的符号 **Ri 和 Rn:的说明**

在分类介绍各类指令之前，先对描述指令的一些符号意义进行一些简单约定：

（1）Ri 和 Rn：R 表示当前工作寄存器区中的工作寄存器，i 表示 0 或 1，即 R0 和 R1。 n 表示 0～7，即 R0～R7，当前工作寄存器的选定是由 PSW 的 RS1 和 RS0 位决定的。

（2）#data：＃表示立即数，data 为 8 位常数。 #data 是指包含在指令中的 8 位立即数。

（3）#data16：包含在指令中的 16 位立即数。

（4）rel：相对地址，以补码形式表示的地址偏移量，范围为-128～+127，主要用于无条件相对短转移指令 SJMP 和所有的条件转移指令中。

（5）addr16：16 位目的地址。 目的地址可在全部程序存储器的 64 KB 空间范围内，主要用于无条件长转移指令 LJMP 和子程序长调用指令 LCALL 中。

（6）addr11：11 位目的地址。目的地址应与下条指令处于相同的 2 KB 程序存储器地址空间范围内，主要用于绝对转移指令 AJMP 和子程序绝对调用指令 ACALL 指令中。

（7）direct：表示直接寻址的地址，即 8 位内部数据存储器 RAM 的单元地址（0～127/255），或特殊功能寄存器 SFR 的地址。 对于 SFR 可直接用其名称来代替其直接地址。

（8）bit：内部数据存储器 RAM 和特殊功能寄存器 SFR 中的可直接寻址位地址。

(9) @：间接寻址寄存器或基地址寄存器的前缀，如@Ri，@DPTR，表示寄存器间接寻址。

(10) (X)：表示 X 中的内容。

(11) ((X))：表示由 X 寻址的单元中的内容，即(X)作地址，该地址的内容用((X))表示。

(12) / 和→符号：/表示对该位操作数取反，但不影响该位的原值。 →表示指令操作流程，将箭头一方的内容，送入箭头另一方的单元中去。

说明：凡指令表上标明符号的地方，在使用时必须根据符号要求，选用具体数值，不能直接写成上述符号。例如不能有 MOV A, Rn 这种写法。

项目 3.2 寻址方式

寻址方式是指某一个 CPU 指令系统中规定的寻找操作数所在地址的方式，或者说通过什么样的方式找到操作数。比如完成 3+2=5 简单运算，在计算机中加数和被加数放在什么地方？CPU 如何得到它们？运算结果存放在什么地方？这些就是所谓的寻址问题。实际上计算机执行程序的过程就是不断地寻找操作数并进行操作的过程。

在计算机中在用汇编语言编程时，数据的存放、传送、运算都要通过指令来完成。

3.2.1 寻址范围

寻址范围就是给某个存储器或者寄存器划分一个区域，如 58H 等，方便指令执行的时候能准确地找到它的位置。因为 8051 的 PC 是 16 位的，DPTR 也是 16 位的，所以寻址空间最大为 0xFFFF，16 位数最大寻址范围就是 64K 也就是 2 的 16 次方。

3.2.2 立即寻址

在这种寻址方式中，操作数直接出现在指令中，它紧跟在操作码的后面，作为指令的一部分与操作码一起存放在程序存储器内，可以立即得到并执行，不需要另去寄存器或存储器等处寻找和取数，故称为立即寻址。该操作数称为立即数，并在其前冠以“#”号作前缀，以表示并非地址。立即数可以是 8 位或 16 位，用十六进制数表示。

例如：MOV A, #5BH ; (A)←5BH

该指令的功能是将立即数 5BH 传送到累加器 A 中，对应的机器码为 74H。 它隐含了寄存器寻址累加器 A 方式，为一个字节，占用一个存储单元；立即数 5BH 紧跟在操作码之后，成为指令代码的一部分，长也是一个字节，占用紧跟在后面的另一个存储单元。 故该指令为双字节指令，其机器码为 74H 5BH。

3.2.3 直接寻址

在指令中直接给出操作数的地址，这种寻址方式就属于直接寻址方式。在这种方式中，指令的操作数部分直接是操作数的地址。直接寻址的地址单元（直接地址）取值必须在 00H-FFH 之间。

例如：MOV A, 40H ; 机器码为 E5H 40H ◆

在 MCS -51 单片机指令系统中，直接寻址方式中可以访问 以下存储器空间：◆

(1) 内部数据存储器的低 128 个字节单元（00H～7FH），在指令中直接以单元地址形式给出。

(2) 特殊功能寄存器。 特殊功能寄存器只能用直接寻址方式进行访问。对特殊功能寄存器直接寻址可用字节为地址，也可用特殊功能寄存器名 。如 MOV A, PSW 与 MOV A, 0D0H 两条指令的作用是一样的，但使用前者更容易理解和阅读。

3.2.4 寄存器寻址

选定某寄存器，自该寄存器中读取或存放操作数，以完成指令规定的操作，称为寄存器寻址。在该寻址方式中，操作数中有一个是寄存器。寄存器一般指 8 个工作寄存器 R0～R7，累加器 A，数据指针 DPTR 和布尔处理器的位累加器 C。实际上寄存器寻址也可以看作是一种直接寻址。

例如：MOV A, R0 ; (A)←(R0)

该指令的功能是把工作寄存器 R0 中的内容传送到累加器 A 中，如：R0 内容为 FFH，则执该指令后 A 的内容也为 FFH。 在该条指令中，源操作数和目的操作数是由寻址 R0 和 A 寄存器得到的，故属于寄存器寻址。 该指令为单字节指令，机器代码为 E8H。

寄存器寻址方式可以访问的存储空间：

（1）4 组工作寄存器 R0～R7 共 32 个工作寄存器，由程序状态字 PSW 中的 RS1、RS0 两位状态来进行当前寄存器组的选择；

（2）特殊功能寄存器 A、B、DPTR。

注意：在 MCS-51 指令系统中，累加器 A 有 3 中不同的表达方式，即 A、ACC 和 0E0H，分属于不同的寻址方式，但指令的执行结果完全相同。寄存器 B 只有在执行乘除指令时才是寄存器寻址方式。

3.2.5 寄存器间接寻址

寄存器间接寻址就是所要查找的操作数位于以寄存器的内容为地址的单元中。这种寻址方式相当于两次寻址。间接寻址的存储空间包括内部数据 RAM 和外部数据 RAM。能用于寄存器间接寻址的寄存器有 R0, R1, DPTR，SP, SP 仅用于堆栈操作。寄存器间接寻址只能使用寄存器 R0, R1 作为地址指针，并在寄存器前面加“@”标志。寻址内部 RAM 区（00～7FH）的 128 个单元，不能访问特殊功能寄存器 SFR。当访问外部 RAM 时，可使用 R0, R1 及 DPTR 作为地址指针。

与寄存器寻址相比，寄存器间接寻址时寄存器存放的是操作数所在的地址，寄存器寻址时寄存器中存放的是操作数。

注意： 寄存器间接寻址方式中，寄存器的内容不是操作数本身，而是操作数地址。寄存器间接寻址符号为“@”，指令 MOV @R0,A 的寄存器间接寻址如图 3.1 所示。

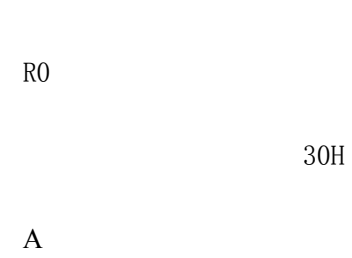


图 3.1　寄存器间接寻址示意图

3.2.6 变址寻址（基址寄存器+变址寄存器间接寻址）

这种寻址方式用于访问程序存储器中的数据表格，它以基址寄存器 DPTR 或 PC 的内容为基本地址，加上变址寄存器 A 的内容作为操作数的地址。两者的内容相加形成 16 位程序存储器地址，该地址就是操作数所在地址。变址寻址。由于程序存储器是只读，因此变址寻址只有读操作而无写操作，在指令符号上采用 MOVC 的形式。指令 MOVC A，@A+DPTR 的变址寻址如图 3.2 所示。

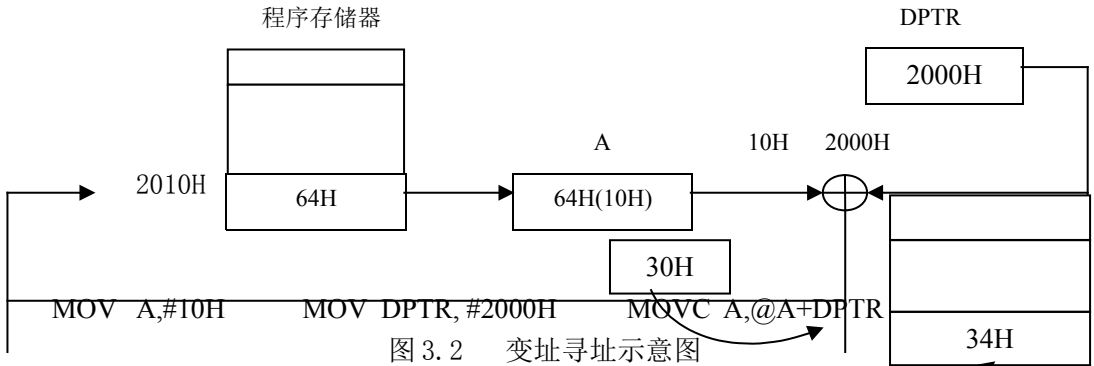


图 3.2　变址寻址示意图

该条指令的功能是把累加器 A 中的内容 10H 与数据指针 DPTR 中的内容 20[illegible]作数地址 2010H，将 2010 中的内容 64H 送入 A，最终 A 的内容为

3.2.7 相对寻址

相对寻址是以当前程序计数器 PC 值加上指令规定的偏移量 rel ， 而构成实际操作数地址的寻址方法。它用于访问程序存储器，只出现在转移指令中用于程序控制。这里的“寻址”不是寻找操作数的地址，而是要得到程序跳转位置对应的目标地址。

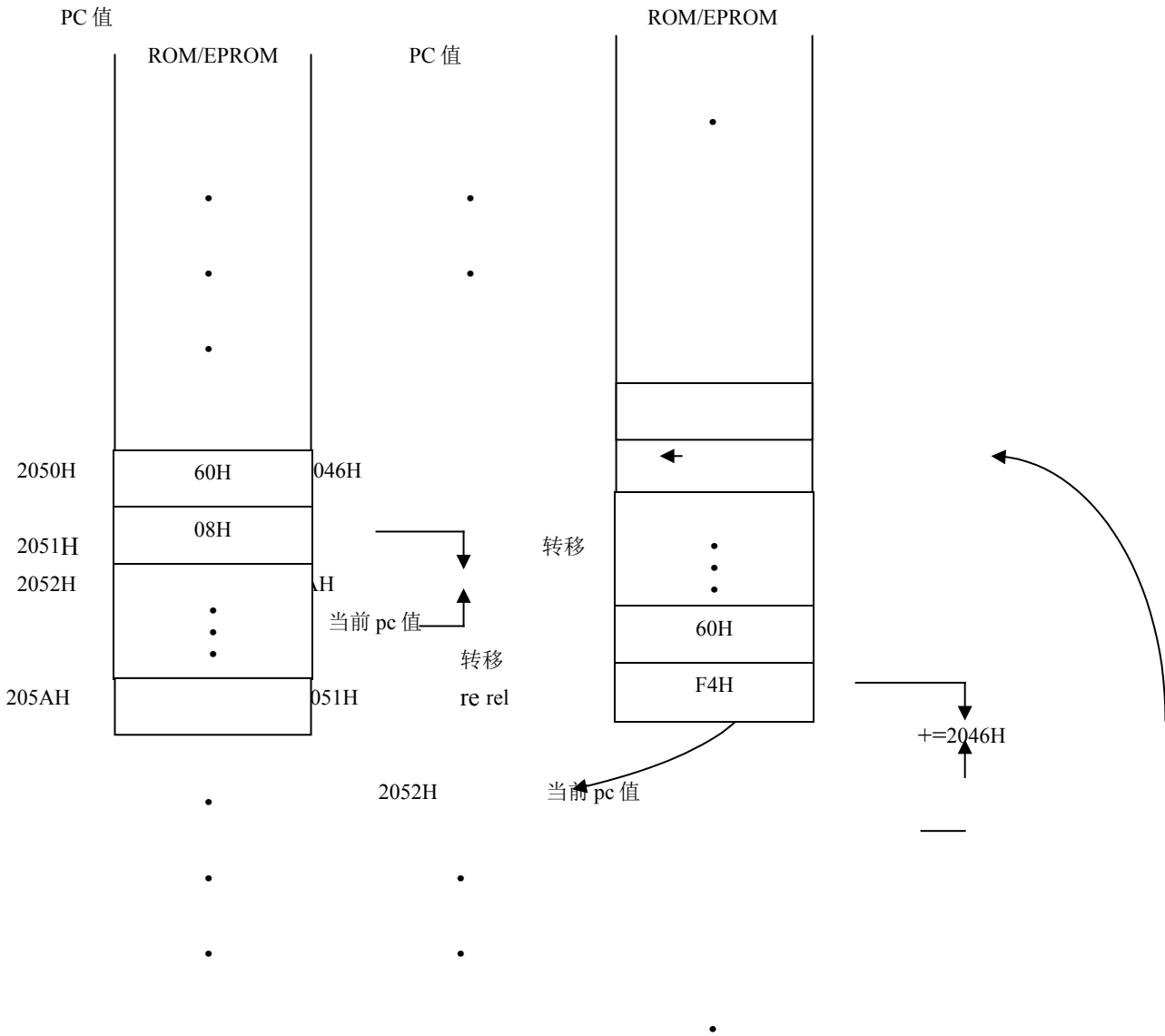
在使用相对寻址时要注意以下两点:

第一、当前 PC 值是指相对转移指令所在地址（一般称为源地址）加上转移指令字节数。即：当前

PC 值 = 源地址 + 转移指令字节数。

例如：JZ rel 是一条累加器 A 为零就转移的双字节指令。 若该指令地址（源地址）为 1030H，则执行该指令时的当前 PC 值即为 1032H。

第二、 偏移量 rel 是有符号的单字节数，以补码表示，程序的转移范围为以 PC 当前值为中心的-128～+127（即 00H～FFH）之间，负数表示从当前地址向上转移，正数表示从当前地址向下转移。 所以，相对转移指令满足条件后，转移的地址（一般称为目的地址）应为：目的地址 = 当前 PC 值 + rel = 源地址 + 转移指令字节数 + rel 例如： 指令 JNZ 08H 和 JNZ F4H 表示累加器 A 为零条件满足后，从源地址（2050H）分别向下、向上转移 10 个单元。其相对寻址示意如图 3.3(a)、(b)所示。这两条指令均为双字节指令，机器代码分别为：60H 08H 和 60H F4H。



（a）指令 JNZ 08H 寻址示意图　　（ b）指令 JNZ F4H 寻址示意图

图 3.3 相对寻址示意图

3.2.8 位寻址

位寻址是对位地址中的内容作位操作的寻址方式。位寻址类似于直接寻址，由指令给出位地址，操作数位于位地址中，而直接寻址给出的是字节地址。两者虽然都是用两位十六进制数表示，但在指令中是可以区分的。

例如：　MOV　A, 20H

　　MOV　C, 20H

两条指令中的“20H”表达了不同的地址，第一条指令中的“20H”代表另一个字节地址，它的寻址方式为直接寻址；而第二条指令中的“20H”代表了一个位地址，采用的寻址方式为位寻址。在“MOV A, 20H”指令中用 C 表示进位位 CY，C 在位操作指令中的作用与 A 类似，为位操作时的累加器。

位寻址可访问的存数空间为：片内 RAM 的 20H～2FH 共 128 位，SFR 中 12 个字节地址能被 8 整除的特殊功能寄存器中的 83 位。

这些位地址在指令中有 4 中表达方式：

（1）直接使用位地址：如 MOV　C,00H；

（2）单元地址加位的表示：如 MOV C, 20H.0；

（3）位名称表示：如 MOV C, OV；

（4）特殊功能寄存器名加位的表示：如 MOV C, PSW.2。

综上所述，在 MCS—51 系列单片机的存储空间中，指令究竟对哪个存储器空间进行操作是由指令操作码和寻址方式确定的。 7 种寻址方式如表 3.1 所示。

表 3.1　7 种寻址方式及使用空间

序号	寻址方式	使用变量	寻址空间
1	立即寻址	#data	程序存储器（指令的常数部分）
2	直接寻址	direct	片内 RAM 低 128B，特殊功能寄存器（SFR）
3	寄存器寻址	Rn, A, B, DPTR	工作寄存器 R0、R7、A、B、DPTR
4	寄存器间接寻址	@Ri, @DPTR	片内 RAM 低 128B，片外 RAM
5	变址寻址	@A+PC, @A+DPTR	程序存储器（数据表）
6	相对寻址	PC+ rel	程序存储器 256B 范围
7	位寻址	C, bit	片内 RAM 的 20H～2FH，特殊功能寄存器可寻址位（字节地址能被 8 整除的 SFR 中的各位）

这 7 种寻址方式中，前 4 种是最常见的。而变址寻址与相对寻址方式在这里了解一下，在后面学习具体的指令时，再加以深入。

项目 3.3　指令系统

3.3.1 数据转送类指令

数据传送类指令共 29 条，它是指令系统中最活跃、 使用最多的一类指令。 一般的操作是把源操作数传送到目的操作数，即指令执行后目的操作数改为源操作数，而源操作数保持不变。 若要求在进行数据传送时，不丢失目的操作数，则可以用交换型传送指令。

数据传送类指令不影响进位标志 CY、半进位标志 AC 和溢出标志 OV，但当传送或交换数据后影响累加器 A 的值时，奇偶标志 P 的值则按 A 的值重新设定。

按数据传送类指令的操作方式，又可把传送类指令分为 3 种类型：数据传送、数据交换和堆栈操

作，并使用 8 种助记符：MOV、MOVX、MOVC、XCH、XCHD、SWAP、PUSH 及 POP。表 3-2 给出了各种数据传送指令的操作码助记符和对应的操作数。

表 3.2 数据传送类指令助记符与操作

功能		助记符	操作数与传送方向
数据传送	内部数据存储器间传送	MOV	A， Rn，@Ri，direct←→#data DPTR←→#data16 A←→Rn，@Ri，direct direct←→direct、Rn、@Ri
	外部数据存储器传送	MOVX	A←→@Ri，@DPTR
	程序存储器传送	MOVC	A←→@A+DPTR，@A+PC
数据交换	字符交换	XCH	A←→Rn，@Ri，direct
	半字节交换	XCHD	A 高四位←→@Ri 低四位
	A 高低 4 位互换	SWAP	A 高四位←→A 低四位
栈操作	压入堆栈	PUSH	SP←→direct
	弹出堆栈	POP	

（一）数据传送指令

1. 内部数据存储器间数据传送指令 MOV

指令格式：MOV [目的操作数]，[源操作数]

内部数据存储器 RAM 区是数据传送最活跃的区域, 可用的指令数也最多, 共有 16 条指令, 指令操作码助记符为 MOV。 内部 RAM 之间源操作数传递关系如图 3.4 所示。 为了便于理解指令功能, 我们按源操作数的寻址方式逐一介绍各条指令。

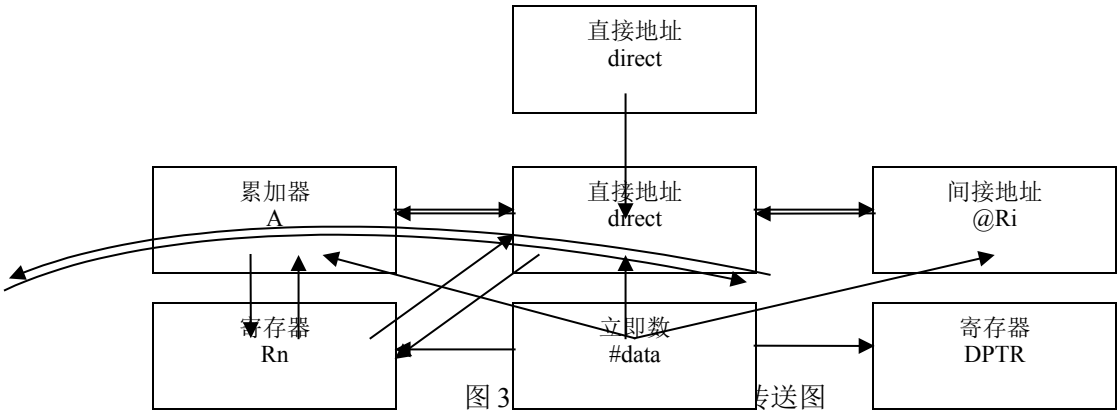


图 3 传送图

（1）以 A 为目的操作数

MOV A, Rn ; A← （Rn）注意： n=0~7

MOV A, direct ; A←（direct）注意： i=0,1

MOV A, @Ri ; A←（（Ri））注意： direct 为 8 位的内部 RAM 或 SFR 的地址（00～FF）

MOV A, #data ; A← #data 注意：data 为 8 位的立即数

这组指令功能为把源操作数的内容送入累加器 A，源操作数有寄存器寻址，直接寻址，间接寻址和立即四种方式。

例如：MOV A, 70H ; A ←(70H) 直接寻址

MOV A, @R0 ; A ←((R0)) 间接寻址

MOV A, R6 ; A ←(R0) 立即寻址

（2）以 Rn 为目的操作数

MOV Rn, A ; Rn ← A

MOV Rn, direct ; Rn ←（direct）

MOV Rn, #data ; Rn ← #data

这组指令的功能是把原操作数的内容送入当前工作寄存器区 R0~R7 中的某一个寄存器。

（3）以直接地址为目的操作数

MOV direct, A ; (direct) ← (A)

MOV direct, Rn ; (direct) ← (Rn)

MOV direct, @Ri◆ ; (direct) ← ((Ri))

MOV direct, #data◆ ; (direct) ← #data

MOV direct2, direct1◆ ; (direct2) ← (direct1)

这组指令的功能是把源操作数的内容送入直接地址指出的存储单元。再次强调 direct 指的是 8 位的内部 RAM 的地址 SFR 的地址（00～FF）

（4）以间接地址为目的的操作数

MOV @Ri, A◆ ;（Ri）← A

MOV @Ri, direct◆ ;（Ri）← direct◆

MOV @Ri, #data◆ ;（Ri）← #data

这组指令的功能是把原操作数的内容送 R0 或 R1 指出的存储单元中去。

例如：设（30H）＝6FH，R1＝40H，执行 MOV @R1, 30H 后，30H 单元中数据取出送入 R1 间接寻址的 40H 单元，(40H)=6FH。

（5）以 DPTR 为目的操作数

MOV DPTR, #data16 ; DPTR ← #data16，

这组指令的功能是将外部存储器某单元地址送到数据指针寄存器 DPTR 中，即把一个 16 位常数送入 DPTR,这是整个指令系统中唯一的一条 16 位数据的传送指令,用来设置地址指针。地址指针 DPTR 由 DPH 和 DPL 组成，这条指令的执行结果是把 8 位立即数送入 DPH，低 8 位立即数送入 DPL。

例如：执行 MOV DPTR， #2000H 后，

（DPTR）= 2000H，（DPL）=00H，（DPH）=20H。

需要说明的是,对于所有 MOV 类指令，A 是一个特别重要的 8 位寄存器,CPU 对它具有其它寄存器所没有的操作指令，加、减、乘、除指令都是以 A 作为操作数进行的，Rn 为 CPU 当前选择的寄存器组中的 R0~R7；直接地址指出的存储单元为内部数据存储器 RAM 的 00H～7FH 和特殊功能寄存器 SFR（地址单元范围为 80H～FFH）；在间接地址中，用 R0 或 R1 作地址指针，访问内部 RAM 的 00H～7FH128 个单元。

2. 外部数据存储器数据传送指令 MOVX

MCS—51 单片机 CPU 对片外扩展的数据存储器 RAM 或 I/O 口进行数据传送，必须采用寄存器间接寻址的方法，通过累加器 A 来完成。一般数据的传送是通过 P0 口和 P2 口完成的，即片外 RAM 地址总线低 8 位由 P0 口送出, 高 8 位由 P2 口送出， 数据总线（8 位）也由 P0 口传送（双向）， 但与低 8 位地址总线是分时传送的。这类数据传送指令共有以下 4 条单字节指令，指令操作码助记符标志为 MOVX。

MOVX A, @DPTR ; (A) ← ((DPTR))

MOVX A, @Ri ; (A) ← ((Ri))

MOVX @DPTR, A ; ((DPTR)) ← (A)

MOVX @Ri, A ; ((Ri)) ← (A)

这组指令中，DPTR 所包含的 16 位地址信息由 P0（低 8 位）和 P2 (高 8 位)输出,而数据信息由 P0 口传送，P0 口作分时复用的总线，由 Ri 作为间接寻址寄存器时，使用 Ri（8 位）作地址指针，能访问的外部数据 RAM 的 00H～FFH256 个单元，P0 口上分时输出 Ri 指定的 8 位地址信息及传输 8 位数据。前两条指令执行时，P3.7 引脚上输出 RD 有效信号，用作外部数据存储器的读选通信号；后两条指令执行时，P3.6 引脚上输出 WR 有效信号，用作外部数据存储器写选通信号。

例如：设（R1）=43H，（R0）=21H，片外 RAM（43H）=65H，执行指令

MOVX A, @R1

MOVX @R0, A

结果为（A）=65H，片外 RAM 21H 单元内容为 65H。

又如：将片外 RAM 2100H 单元内容送到片外 3000H 单元中

```
MOV   DPTR, #2100H
MOVX  A, @DPTR
MOV   DPTR, #3000H
MOVX  @DPTR, A
```

注意： ① 片内和片外数据传送用的助记符是不一样的，MOV 用于片内 RAM，MOVX 用于片外 RAM。

② 片外数据传送与累加器 A 有关。

③ 若片外 RAM 的地址小于 256B，常采用@Ri；若片外 RAM 的地址大于 256B，则常采用@DPTR。

④ MOVX 常用于对外部数据存储器及外围设备的读写操作。

3. 程序存储器传送指令 MOVC

程序存储器包括：片内程序存储器和片外程序存储器。

由于对程序存储器只能读而不能写，因此数据传送是单向的，即只能从程序存储器读数据，并送到累加器 A 中。这类指令有以下两条，指令操作码助记符为 MOVC。

MOVC A, @A+DPTR ； (A) ← ((A)+(DPTR))

MOVC A, @A+PC ； (PC) ← (PC)+1, (A) ← ((A)+(PC))

这组指令以 PC（或 DPTR）做基址寄存器，累加器 A 的内容作为无符号整数和 PC（或 DPTR ）的内容相加后得到一个 16 位的地址，把由该地址指出的程序存储单元的内容送到累加器 A。需要特别注意的是 PC 的内容是只该条指令执行后的下一条指令的 PC 值。

例如：已知（A）＝30H，执行地址 1000H 处的指令：

1000H： MOVC A， @A+PC

该指令为单字节指令，本身占用一个单元，下一条指令的地址为 1001H，（PC）＝1001H，再加上 A 中的 30H，得 1031H，结果为将程序存储器 1031 中的内容送入 A。

又如：（A）＝30H，（DPTR）＝3000H，程序存储器单元（3030H）＝50H，执行 MOVC A，@A+DPTR 后，（A）＝50H。

这是两条很有用的查表指令，可用来查找存放在外部程序存储器中的常数表格。上述第一条指令的优点是不改变特殊功能寄存器和 PC 的状态，只要根据 A 的内容就可以取出表格中的常数。缺点是表格只能放在该条查表指令后面的 256 个单元之中，表格的大小受到限制，而且表格只能被一段程序所利用。

第二条指令是以 DPTR 作为基址寄存器，累加器 A 的内容作为无符号数与 DPTR 内容相加，得到一个 16 位的地址，并把该地址指出的程序存储器单元的内容送到累加器 A。这条指令的执行结果只与指针 DPTR 及累加器 A 的内容有关， 与该指令存放的地址无关。因此，表格的大小和位置可以在 64 KB 程序存储器中任意安排，并且一个表格可以为各个程序块所共用。

例 1：在外部 ROM / EPROM 中，从 2000H 单元开始依次存放 0～9 的平方值：0、1、2、4、9、……、81，要求依据累加器 A 的值（0～9）来查找所对应的平方值，分析下述程序的结果。

```
MOV   DPTR, #2000H    ; (DPTR) ← 2000H
MOV   A, #09H         ; (A) ← 0FH
MOVC  A, @A+DPTR      ; (A) ← ((A)+(DPTR))
```

例 2： 仍以例 1 外部 ROM/EPROM 2000H 单元开始存放 0～9 的平方值, 以 PC 作为基址寄存器进行查表。

解：设 MOVC 指令所在地址(PC)=1FF0H, 则

偏移量 = 2000H－(1FF0H+1) = 0FH◆

相应的程序如下：

```
MOV   A, #09H         ; (A) ← 09H
ADD   A, #0FH         ; 地址调整
MOVC  A, @A+PC        ; (A) ← ((A)+(PC)+1)
```

执行结果为：(PC)=1FF1H，(A)=51H。

（二）数据交换指令

数据交换主要是在片内 RAM 单元与累加器 A 之间进行，分整字节和半字节两种。

（1）字节交换指令 XCH

```
XCH    A, direct      ; (A) ← (direct)
XCH    A, @Ri         ; (A) ← ((Ri))
XCH    A, Rn          ; (A) ← (Rn)
```

这组指令的功能是将累加器 A 的内容和原操作数的内容相互交换。

（2）半字节交换

```
XCHD  A,  @Ri◆          ; (A3～0) ← ((Ri)3～0)
```

这组指令的功能是将累加器 A 的低 4 位内容和内部 RAM 的低 4 位内容互相交换。

（3）累加器高低半字节交换指令

```
SWAP  A                 ; (A7～4) ← (A3～0)
```

例如：设（R0）＝30H，（30H）＝4AH，（A）＝28H 则

```
执行      XCH   A, @R0      ; 结果为：（A）＝4AH，（30H）＝28H
          XCHD  A, @R0      ; 结果为：（A）＝2AH，（30H）＝48H
          SWAP  A           ; 结果为：（A）＝82H
```

（三）堆栈操作指令

在 MCS-51 单片机中，堆栈区开辟在片内 RAM 的 30H～7FH 之间，堆栈的操作遵循“先进后出”的原则，由堆栈指针 SP 自动跟踪栈顶地址。单片机堆栈编址采用向上生产方式，即栈底占用较低地址，栈顶占用较高地址。数据写入堆栈称为入栈，数据从堆栈读出称为出栈。

（1）入栈指令 PUSH

```
PUSH    direct ;          (SP)←(SP)+1;    ((SP))←(direct)
```

功能： 把堆栈指针加 1 后，将直接地址单元的内容送进栈顶单元，原直接地址单元内容不变。

该指令的操作数只有一个为源操作数，目的操作数已默认为堆栈区的栈顶位置。

例如：设（SP）=60H,（A）=30H,（B）=70H,执行下列指令

```
PUSH   A
PUSH   B
```

结果为（61H）=30H，（62H）=70H，（SP）=62H。

（2）出栈指令 POP

```
POP   direct ;          (direct)←((SP)) ;   (SP)←(SP)-1
```

功能：将栈顶单元的内容传送给直接地址单元后，SP 内容减 1。

例如：设（SP）=62H,（62H）=70H，（61H）=30H，执行下列指令

```
POP   DPH
POP   DPL
```

结果为（DPTR）=7030H，（SP）=60H.

例 3：若在外部 ROM/EPROM 中 2000H 单元开始依次存放 0～9 的平方值, 数据指针(DPTR)=3A00H, 用查表指令取出 2003H 单元的数据后, 要求保持 DPTR 中的内容不变。 完成以上功能的程序如下：

```
MOV    A, ＃03H          ; (A) ← 03H
PUSH   DPH               ; 保护 DPTR 高 8 位入栈
PUSH   DPL               ; 保护 DPTR 低 8 位入栈
MOV    DPTR , ＃2000H    ; (DPTR) ← 2000H
MOVC   A, @A+DPTR        ; (A) ← (2000H+03H)
POP    DPL               ; 弹出 DPTR 低 8 位
POP    DPH               ; 弹出 DPTR 高 8 位,（先进后出）
```

例 4：将片内 RAM 30H 单元与 40H 单元中的内容互换。

方法 1（直接地址传送法）：

```
MOV 31H , 30H
MOV 30H , 40H
MOV 40H , 31H
SJMP $
```

方法 2（间接地址传送法）：

```
MOV   R0 , #40H
MOV   R1 , #30H
MOV   A , @R0
MOV   B , @R1
MOV   @R1 , A
MOV   @R0 , B
SJMP  $
```

方法 3（字节交换传送法）：

```
MOV A,30H
XCH  A，40H
MOV 30H，A
SJMP $
```

方法 4（堆栈传送法）：

```
PUSH    30H
PUSH    40H
POP30H
POP     40H
SJMP  $
```

例 5：将片内 RAM 从 20H 开始的 10 个单元内的数据传递给片内 RAM 30H 开始的 10 个单元。

```
        ORG   1000H
        MOV   R7, #0AH
        MOV   R0, #20H
        MOV   R1, #30H
LOOP:   MOV   A, @R0
        MOV   @R1, A
        INC    R0
        INC    R1
        DJNZ   R7, LOOP
        SJMP   $
       END
```

3.3.2 算术运算类指令

算术运算类指令共有 24 条， 如下表 3.3 所示。可分为加法、带进位加法、带借位减法、加 1 减 1，乘除及十进制调整指令共 6 组。它主要完成加、减、乘、除四则运算，以及增量、减量和二—十进制调整操作， 对 8 位无符号数可进行直接运算；借助溢出标志， 可对带符号数进行 2 的补码运算； 借助进位标志，可进行多字节加减运算， 也可以对压缩 BCD 码（即单字节中存放两位 BCD 码）进行运算。

表 3.3 算术运算指令

指令助记符	功能简述	字节数	振荡器周期数
ADD A，Rn	A←(A)+(Rn)	1	12
ADD A，direct	A←(A)+(direct)	2	12
ADD A，@Ri	A←(Ri)+(A)	1	12
ADD A，#data	A←(A)+data	2	12
ADDC A, Rn	A←(A)+(Rn)+CY	1	12
ADDC A, direct	A←(A)+(direct)+CY	2	12
ADDC A, @Ri	A←(A)+(Ri)+CY	1	12
ADDC A, #data	A←(A)+data+CY	2	12
INC A	A←(A)+1	1	12
INC Rn	Rn←(Rn)+1	1	12

INC @Ri	Ri←(Ri)+1	1	12
INC direct	direct←(direct)+1	1	12
INC DPTR	DPTR←(DPTR)+1	1	24
DA A	对 A 进行十进制调整	1	12
SUBB A，Rn	A←(A)-(Rn)-CY	1	12
SUBB A，@Ri	A←(A)-(Ri)-CY	1	12
SUBB A，direct	A←(A)-direct-CY	2	12
SUBB A，#data	A←(A)-data-CY	2	12
DEC A	A←(A)-1	1	12
DEC Rn	Rn←(Rn)-1	1	12
DEC direct	direct←(direct)-1	2	12
DEC @Ri	Ri←(Ri)-1	1	12
MUL AB	AB←(A)*(B)	1	48
DIV AB	AB←(A)/(B)	1	48

（一）加法指令

1. 不带 Cy 位加法指令 ADD

加法指令共有如下 4 条指令, 操作数助记符为 ADD。

ADD A, ＃data ; (A) ← (A)+＃data

ADD A, direct ; (A) ← (A)+(direct)

ADD A, @Ri ; (A) ← (A)+((Ri))

ADD A, Rn ; (A) ← (A)+(Rn)

例如：设（A）=84H，（30H）=8DH，（PSW）=00H，执行指令

ADD A, 30H

试分析运算结果及对各标志位的影响。

将 A 中的内容与 30H 中的内容相加，即

（A）= 1 0 0 0 0 1 0 0

+（30H）= 1 0 0 0 1 1 0 1

（A）= 1 0 0 0 1 0 0 0 1

则运算结构为（A）=11H，（PSW）=0C4H，其中（CY）=1，(AC)=1，（OV）=1，（P）=0。

2. 带进位加法指令 ADDC

带进位加法指令有如下 4 条指令, 其助记符为 ADDC。

ADDC A, ＃data ; (A) ← (A)+(CY)+＃data

ADDC A, direct ; (A) ← (A)+(CY)+(direct)

ADDC A, @Ri ; (A) ← (A)+(CY)+((Ri))

ADDC A, Rn ; (A) ← (A)+(CY)+(Rn)

从上述两组加法指令中可看出，加法指令的一个原操作数总是 A，且运算结果也放在 A 中。两个操作数可以同时是无符号数或是有符号数。家法指令影响 PSW 中的 Cy、AC、OV 及 P 位。带进位加法指令的功能与普通加法之类似，唯一的不同之处是，在执行带进位加法时，还要将上一次进位标志 Cy 的内容也一起加进去，对于标志位的影响也与普通加法指令相同。

例如：设（A）=42H，（R3）=68H，（PSW）=80H，执行指令

ADDC A, R3

将 A 中的内容与 R3 中的内容相加，并考虑当前的进位值，即

（A）= 0 1 0 0 0 0 1 0

（30H）= 0 1 1 0 1 0 0 0

+(CY）= 1

（A）= 1 0 1 0 1 0 1 1

则运算结构为（A）=0ABH，（PSW）=05H，其中（CY）=0，(AC)=0，（OV）=1，（P）=1。

3. 加 1 指令 INC

```
INC  A◆  ;  （A）←（A）+1
INC  Rn◆  ;  （direct）←  (direct)+1
INC  direct  ;◆ ((Ri)) ←  ((Ri)) +1
INC  @Ri◆ ;  (Rn) ← (Rn)+1
INC  DPTR◆;  (DPTR) ← (DPTR)+1
```

这组指令的功能是：将指令中所指出操作数的内容加 1。若原来的内容为 0FFH, 则加 1 后将产生溢出, 使操作数的内容变成 00H, 但不影响任何标志。最后一条指令是对 16 位的数据指针寄存器 DPTR 执行加 1 操作, 指令执行时, 先对低 8 位指针 DPL 的内容加 1, 当产生溢出时就对高 8 位指针 DPH 加 1, 但不影响任何标志。

例如：(30H)=22H，执行 INC 30H 后，（30H）＝23H。

又如：设（R0）＝7EH，（7EH）＝FFH，（7FH）＝38H,（DPTR）＝10FEH，分析逐条执行下列指令后各单元的内容。

```
INC  @R0          ; 使 7EH 单元内容由 FFH 变为 00H
INC  R0           ; 使 R0 单元内容由 7EH 变为 7FH
INC  @R0          ; 使 7FH 单元内容由 38H 变为 39H
INC  DPTR         ; 使 DPL 为 00H ，DPH 不变
INC  DPTR         ; 使 DPL 为 00H ， DPH 为 11H
INC  DPTR         ; 使 DPL 为 01H ，DPH 不变
```

4. 十进制调整指令 DA

```
DA   A            ；把 A 中按二进制相加的结果调整成按 BCD 码相加的结果
```

这条指令对累加器 A 参与的 BCD 码加法运算所获得的 8 位结果进行十进制调整，使累加器 A 的内容调整为二位压缩性 BCD 码的数。使用时必须注意,它只能跟在加法指令之后，不能对减法指令的结果进行调整，且其结果不影响溢出标志为。执行该指令时，判断 A 中的低 4 位是否大于 9 和辅助进位标志 AC 是否为“1”，若两者有一个条件满足，则低 4 位加 6 操作，同样，A 中的高 4 位大于 9 或进位标志 Cy 为“1”两者有一个条件满足时，高 4 位加 6 操作。

例如：设（A）=42H，表示十进制数 42 的压缩 BCD 码。（R3）=68H,表示十进制数 68H 的压缩，（PSW）=80H，执行指令：

```
ADDC  A,  R3
DA    A
```

将 A 中的内容与 R3 中的内容进行带进位位加法运算，并根据结果进行调整，即

```
（A）= 0 1 0 0  0 0 1 0  （42 的 BCD 码）
（R3）= 0 1 1 0  1 0 0 0  （68 的 BCD 码）
+（CY）=          1
（A）= 1 0 1 0  1 0 1 1  （A、B 为十六进制数）
     + 0 1 1 0  0 1 1 0  （66H 为调整值）
       1 0 0 0 1  0 0 0 1  （111 为 BCD 码）
```

执行结果：（A）＝（11）BCD，（CY）＝1 。

（二）减法指令

1. 带借位减法指令 SUBB

带借位减法指令有如下 4 条指令, 其助记符为 SUBB。

```
SUBB  A, ＃data   ; (A) ← (A)－(CY)－＃data
SUBB  A, direct   ; (A) ← (A)－(CY)－(direct)
SUBB  A, @Ri      ; (A) ← (A)－(CY)－((Ri))
```

SUBB A, Rn ; (A) ← (A)－(CY)－(Rn)

这组指令的功能是：将累加器 A 的内容与第二操作数及进位标志相减， 结果送回到累加器 A 中。在执行减法过程中，如果位 7（D7）有借位， 则进位标志 Cy 置“1”， 否则清“0”；如果位 3（D3）有借位，则辅助进位标志 AC 置“1”， 否则清“0”； 如位 6 有借位而位 7 没有借位, 或位 7 有借位而位 6 没有借位，则溢出标志 OV 置“1”，否则清“0”。 若要进行不带借位的减法操作， 则必须先将 Cy 清“0”。

注意：由于减法指令只有带借位减法指令， 因此，若要进行不带借位位的减法操作，需先清借位位，即置 CY=0。清 CY 有专门的指令，它属于位操作类指令，指令为：CLR C。

例如：设(A)=52H, (R0)=B4H

执行指令：

CLR C ; (CY)←0

SUBB A, R0 ; (A) ← (A)－(CY)－(R0)

结果为：（A）=9EH, （CY）=1,（AC）=1, （OV）=1,（P）=1。

2. 减 1 指令 DEC

有如下 4 条指令, 助记符为 DEC。

DEC A ; (A) ← (A)－1

DEC direct ; (direct) ← (direct)－1

DEC @Ri ; ((Ri)) ← ((Ri))－1

DEC Rn ; (Rn) ← (Rn)－1

这组指令的功能是：将指出的操作数内容减 1。 如果原来的操作数为 00H, 则减 1 后将产生下溢出，使操作数变成 0FFH，但不影响任何标志。

例如：（R0）＝30H，（30H）＝22H，执行 DEC @R0 后，(30H)=21H。

（三）乘除法指令

乘、 除法指令为单字节 4 周期指令，在指令执行周期中是最长的两条指令。

1. 乘法指令

乘法指令完成单字节的乘法, 只有一条指令： ◆

MUL AB◆ ;BA ←（B）×（A）

这条指令的功能是：将累加器 A 的内容与寄存器 B 的内容相乘， 乘积的低 8 位存放在累加器 A 中, 高 8 位存放于寄存器 B 中。如果乘积超过 0FFH, 则溢出标志 OV 置“1”，否则清“0”。 进位标志 Cy 总是被清“0”。

例如：（A）=30H，（B）=60H，执行 MUL AB 后，（A）=00H，（B）=12H。

又如：若(A)=4EH（78），(B)=5DH（93）

执行指令：MUL AB

结果为：积为(BA)=1C56H（7254) > FFH（255)，(A)=56H，(B)=1CH，（OV）=1，（CY）=0，（P）=0。

另外，乘法指令本身只能进行两个 8 位数的乘法运算，要进行多字节乘法还需编写相应的程序。

2. 除法指令◆

除法指令完成单字节的除法，只有一条指令： ◆

DIV AB◆ ； A←（A）/（B）的商，B←（A）/（B）的余数

这条指令的功能是：将累加器 A 中的内容除以寄存器 B 中的 8 位无符号整数，所得商的整数部分存放在累加器 A 中，余数部分存放在寄存器 B 中，清“0”进位标志 Cy 和溢出标志 OV。 若原来 B 中的内容为 0，则执行该指令后 A 与 B 中的内容不定，并将溢出标志 OV 置“1”， 在任何情况下，进位标志 Cy 总是被清“0”。

若（B）=00H,则指令执行后（OV）=1，A 与 B 不变。

例如，（A）=30H，（B）=07H，执行 DIV AB 后，（A）=06H，（B）=06H。

例 1：将 4 个单字节数放在片内 20H∽23H 单元,它们求和的结果放在片内 30H，31H 单元。

```
        ORG    1000H
        MOV    R7, #04H
        MOV    R0, #20H
        CLR    A
        MOV    31H, A
LOOP:   ADD    A, @R0
        JNC    NEXT
        INC    31H
NEXT:   INC    R0
        DJNZ   R7, LOOP
        MOV    30H, A
        SJMP   $
        END
```

例 2：片内 RAM 中 30H 单元内存有一个二进制数，设计一段程序把这个数转换为 BCD 编码的十进制数，BCD 码（十位和个位）放在累加器 A 中，百位放在 R3 中。

```
        ORG    2200H
        MOV    A, 30H
        MOV    B, #64H
        DIV    AB
        MOV    R3, A
        MOV    A, #0AH
        XCH    A, B
        DIV    AB
        SWAP   A
        ORL    A,B
        SJMP   $
        END
```

3.3.3 逻辑运算类指令

逻辑运算及移位指令共有 24 条，如表 3.4 所示，其中逻辑指令有“与”，“或”、“非”、累加器 A 清零和求反 20 条，移位指令 4 条。

表 3.4 逻辑运算指令

指令助记符	功能简述	字节数	振荡器周期数
CLR A	累加器调零	1	12
CPL A	累加器取反	1	12
RL A	累加器循环左移 1 位	1	12
RLC A	累加器带进位标志位循环左移 1 位	1	12
RR A	累加器循环左移 1 位	1	12
RRC A	累加器带进位标志位循环右移 1 位	1	12
ANL A，Rn	A←(A) ∧（Rn）	1	12
ANL A，direct	A←(A) ∧（direct）	2	12
ANL A，@Ri	A←(A) ∧（Ri）	1	12
ANL A，#data	A←(A) ∧data	2	12
ANL direct，A	direct←(direct) ∧（A）	3	12
ANL direct，#data	direct←(direct) ∧data	3	24

ORL A，Rn	A←(A) ∨(Rn)	1	12
ORL A，Rn	A←(A) ∨(Rn)	1	12
ORL A，direct	A←(A) ∨(direct)	2	12
ORL A，@Ri	A←(A) ∨(Ri)	1	12
ORL A，#data	A←(A) ∨data	2	12
ORL direct，A	direct←(direct) ∨(A)	2	12
ORL direct，#data	direct←(direct) ∨data	3	24
XRL A，Rn	A←(A)⊕(Rn)	1	12
XRL A，direct	A←(A)⊕(direct	2	12
XRL A，@Ri	A←(A)⊕((Ri))	1	12
XRL A，#data	A←(A)⊕data	2	12

续 表

XRL direct，a	direct←(direct)⊕(A)	2	12
XRL direct，#data	direct←(direct)⊕data	3	24

（一） 逻辑“与”运算指令

逻辑“与”运算指令共有如下 6 条, 其助记符为 ANL。

ANL direct, A ; (direct) ← (direct)∧(A)

ANL direct, #data ; (direct) ← (direct)∧＃data

ANL A, #data ; (A) ← (A)∧＃data

ANL A, direct ; (A) ← (A)∧(direct)

ANL A, @Ri ; (A) ← (A)∧((Ri))

ANL A, Rn ; (A) ← (A)∧(Rn)

这组指令的功能是：将两个操作数的内容按位进行逻辑与操作, 并将结果送回目的操作数的单元中。

（二）逻辑“或”运算指令

逻辑“或”运算指令共有如下 6 条指令, 其助记符为 ORL。

ORL direct, A ; (direct) ← (direct)∨(A)

ORL direct, #data ; (direct) ← (direct)∨＃data

ORL A, #data ; (A) ← (A)∨＃data

ORL A, direct ; (A) ← (A)∨(direct)

ORL A, @Ri ; (A) ← (A)∨((Ri))

ORL A, Rn ; (A) ← (A)∨(Rn)

这组指令的功能是： 将两个操作数的内容按位进行逻辑或操作，并将结果送回目的操作数的单元中。

注意：逻辑“与”指令常用于屏蔽（置 0）字节中某些位。若清除某位，则用“0”和该位相与，若保留某位，则用“1”和该位相与。逻辑“或”指令将两个指定的操作数按位进行逻辑“或”操作。 它常用来使字节中某些位置“1”, 欲保留（不变）的位用“0”与该位相或，而欲置位的位则用“1”与该位相或。

例如： (A)=FAH=11111010B， (R1)=7FH=01111111B

执行指令： ANL A, R1 ; (A) ← 11111010∧01111111

结果为： (A)=01111010B=7AH。

又如：根据累加器 A 中 4～0 位的状态， 用逻辑与、 或指令控制 P1 口 4～0 位的状态，P1 口的高 3 位保持不变。

```
ANL   A, ＃00011111B  ; 屏蔽 A 的高 3 位
ANL   P1, ＃11100000B ; 保留 P1 的高 3 位
ORL   P1, A           ; 使 P14～0○按 A4～0○置位
```

若上述程序执行前： (A)=B5H=10110101B，（P1)=6AH=01101010B，则

执行程序后： (A)=15H=00010101B， (P1)=75H=01110101B。

（三）逻辑“异或”运算指令

“异或”运算是当两个操作数不一致时结果为 1，两个操作数一致时结果为 0，这种运算也是按位进行，共有如下 6 条指令，其助记符为 XRL。

```
XRL    direct, A          ; (direct) ← (direct) ⊕(A),
XRL    direct, ＃data      ; (direct) ← (direct) ⊕ ＃data
XRL    A, ＃data          ; (A) ← (A)⊕ ＃data
XRL    A, direct          ; (A) ← (A)⊕(direct)
XRL    A, @Ri             ; (A) ← (A) ⊕((Ri ))
XRL    A, Rn              ; (A) ← (A) ⊕(Rn)
```

这组指令的功能是： 将两个操作数的内容按位进行逻辑异或操作，并将结果送回到目的操作数的单元中。

逻辑“异或”指令常用来对字节中某些位进行取反操作，欲某位取反则该位与“1”相异或；欲某位保留则该位与“0”相异或。 还可利用异或指令对某单元自身异或，以实现清零操作。

例如：若（A）=B5H=10110101B， 执行下列指令：

```
XRL    A, #0F0H        ; A 的高 4 位取反, 低 4 位保留
MOV    30H, A          ; (30H) ← (A) = 45H
XRL    A, 30H          ; 自身异或使 A 清零
```

执行后结果：(A)=00H。

以上逻辑“与”、“或”、“异或”各 6 条指令有如下共同的特点：

(1) 逻辑“与”ANL、“或”ORL、“异或”XRL 运算指令除逻辑操作功能不同外，三者的寻址方式相同， 指令字节数相同， 机器周期数相同。

(2) ANL、ORL、XRL 的前两条指令的目的操作数均为直接地址方式，可很方便地对内部 RAM 的 00H～FFH 任一单元或特殊功能寄存器的指定位进行清零、置位、取反、保持等逻辑操作。

(3) ANL、ORL、XRL 的后 4 条指令，其逻辑运算的目的操作数均在累加器 A 中，且逻辑运算结果保存在 A 中。

（四）累加器 A 清零与取反指令

```
CLR   A    ; (A) ← 00H
CPL   A    ; (A) ←(A )
```

第 1 条是对累加器 A 清零指令， 第 2 条是把累加器 A 的内容取反后再送入 A 中保存的对 A 求反指令， 它们均为单字节指令。 若用其它方法达到清零或取反的目的， 则至少需用双字节指令。

例如： 上例中用异或指令使累加器清零，需要两条双字节指令： MOV 30H, A 和 XRL A, 30H 共占用四字节存储空间；若用 MOV A, ＃00H 实现累加器清零， 也需一条双字节指令，而用 CLR A 一条单字节指令就可完成 A 清零的操作，大大节约了程序的存储空间和程序的执行时间。

（五） 移位指令

移位指令有如下循环左移、 带进位位循环左移、 循环右移和带进位位循环右移 4 条指令, 移位只能对累加器 A 进行。

```
循环左移：          RL    A      ; (A n+1) ← (An) , (A0) ← (A7)
带进位位循环左移 ：  RLC   A      ; (A n+1○) ← (An ), (CY) ← (A7 ),  (A0 ) ← (CY)

循环右移 ：          RR    A      ; (An) ← (A n+1 ),  (A7 ) ← (A0)
带进位位循环右移：    RRC   A      ; (An) ← (A n+1 ○), (CY) ← (A0), (A7) ← (CY)
```

以上移位指令操作, 可用图 3.5 表示。

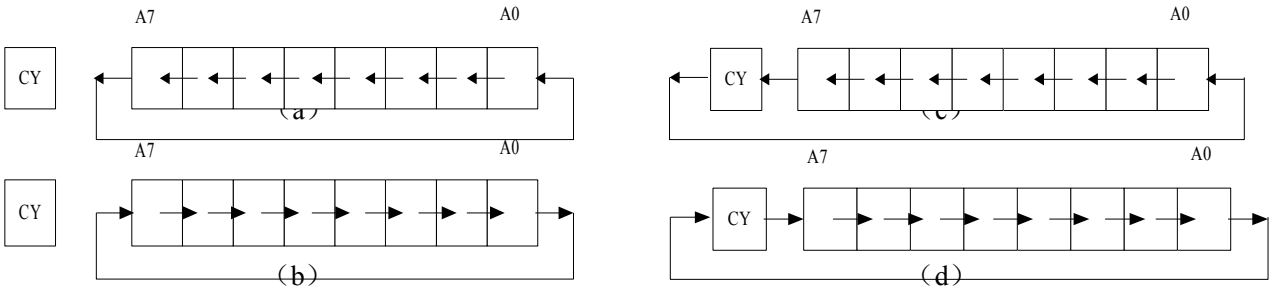


图 3.5 移位指令操作示意图

另外， 值得一提的是在前述数据传送类指令中有一条累加器 A 的内容半字节交换指令：

SWAP A ; (A)7～4 (A)3～0ↄ ，

它实际上相当于执行循环左移指令 4 次， 该指令在 BCD 码的变换中是很有用的。

例如： 设(A)=43H，(CY)=0，则

执行指令： RL A

RLC A

RR A

RRC A

结果为： (A)=86H , (CY)=0

(A)=0CH , (CY)=1

(A)=06H , (CY)=1

(A)=83H , (CY)=0

3.3.4 控制转移类指令

控制转移指令用于改变程序计数器 PC 的值，以控制程序执行走向。因此，其作用区间必然是程序存储器空间。控制转移指令分为无条件转移指令、条件转移指令、调用和返回指令。

控制转移指令共有 17 条，不包括按布尔变量控制程序转移指令（见表 3.5）。其中有 64 KB 范围内的长调用、长转移指令；有 2 KB 范围内的绝对调用和绝对转移指令；有全空间的长相对转移及一页范围内的短相对转移指令；还有多种条件转移指令。由于 MCS-51 提供了较丰富的控制转移指令，因此在编程上相当灵活方便。这类指令用到的助记符共有 10 种：AJMP、LJMP、SJMP、JMP、ACALL、LCALL、JZ、JNZ、CJNE、DJNZ。

表 3.5 控制转移指令

指令助记符	功能简述	字节数	振荡器周期数
AJMP $addr_{11}$	2KB 内绝对转移	2	24
LJMP $addr_{16}$	64KB 内绝对转移	3	24
SJMP rel	相对短转移	2	24
JMP @A+DPTR	相对长转移	1	24
JZ rel	累加器为零转移	2	24
JNZ rel	累加器不为零转移	2	24
CJNE A，direct，rel	A 的内容与直接寻址字节的内容不等转移	3	24

续 表

CJNE A，#data，rel	A 的内容与立即数不等转移	3	24
CJNE Rn，#data，rel	Rn 的内容与立即数不等转移	3	24
CJNE @Rn，#data，rel	内部 RAM 单元的内容与立即数不等转移	3	24
DJNZ Rn，rel	寄存器内容减 1 不为零转移	3	24

DJNZ direct，rel	直接寻址字节内容减 1 不为零转移	3	24
ACALL $addr_{11}$	2KB 内绝对调用	2	24
LCALL $addr_{16}$	64KB 内绝对调用	3	24
RET	子程序返回	1	24
RET1	中断返回	1	24

（一）无条件转移指令（4 条）

1. 绝对转移指令

AJMP　　addr11　　; PC←（PC）+2，　PC 10~0 ← addr11

这是一条二字节指令，其机器码是：

a_{10}	a_9	a_8	0	0	0	0	1	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0

指令提供的 11 位地址中，a7~a0 在第二字节，a10~a8 则占据第一字节的高 3 位，第一个字节的低 5 位固定为 00001，是该指令的操作码。其中，a10~a8 和 a7~a0 由 11 位绝对地址 addr11 填入。

本条指令执行时，先将 PC+2，然后将 addr11 送入 PC10～PC0，而以 PC15~PC11 保持不变，这样得到跳转的目的地址。也就是说，AJMP 指令提供程序转移的目的地址，以指令操作数 addr11 提供的 11 位地址去替换本条指令所在地址加 2 形成的 PC 值的低 11 位地址，得到新的 PC 值，此即转移后的目的地址。需要注意的是，目的地址与 AJMP 后面一条指令的第一个字节必须在同一个 2KB 区域的存储器内，这是由于 AJMP 指令执行时，所在地址高 5 位保持不变这一特点决定的。表 3.6 给处理程序存储器空间 32 个 2K 地址范围。

表 3.6　程序存储器空间 32 个 2K 地址范围

0000H~07FFH	0800H~0FFFH	1000H~17FFH	1800H~1FFFH
2000H~27FFH	2800H~2FFFH	3000H~37FFH	3800H~3FFFH
4000H~47FFH	4800H~4FFFH	5000H~57FFH	5800H~5FFFH
6000H~67FFH	6800H~6FFFH	7000H~77FFH	7800H~7FFFH
8000H~87FFH	8800H~8FFFH	9000H~97FFH	9800H~9FFFH
A000H~A7FFH	A800H~AFFFH	B000H~B7FFH	B800H~BFFFH
C000H~C7FFH	C800H~CFFFH	D000H~D7FFH	D800H~DFFFH
E000H~E7FFH	E800H~EFFFH	F000H~F7FFH	F800H~FFFFH

例如：执行下列指令后新的目标地址是否合理？

（1）0100H　AJMP　0210H

（2）3456H　AJMP　3B10H

分析：（1）PC 当前值=0100H+02H=0102H，高 5 位是 00000B

目标地址=0210H，高 5 位是 00000B

转移前后 PC 值得高 5 位相同，所以目标地址合理。

（2）PC 当前值=3456H+02H=3457H，高 5 位是 00110B

目标地址=3B10H，高 5 位是 00111B

PC 当前值与转移目标地址高 5 位不同，不在同一个 2KB 区域内，目标地址不合理。

2. 短（相对）转移指令

SJMP　rel　　；PC ← （PC）+ 2 + rel

SJMP 指令也叫短跳转指令，它是相对寻址方式转移指令，其中 rel 为相对偏移量。该指令功能是计算目的地址，并按计算得到的目的地址实现程序的相对转移。计算公式：

目的地址＝（PC）+2+rel

式中偏移量 REL 是一个带符号的 8 位二进制补码。

例如：在 835AH 地址由 SJMP 指令

835AH：S JMP 35 H

源地址为 835AH，rel＝35H 是正数，因此程序向前转移。目的地址＝835AH+02H+35H＝8391H。即此条指令执行完毕，程序转到 8391H 地址去执行。

例如：在 835A 地址有 SJMP 指令

835AH： SJMP E7H

rel＝E7H 是负数 19H 的补码，因此程序向后转移。目的地址＝835AH+02H－19H=8343H。即此条指令执行完毕，程序向后转移到 8343H 地址去执行。

Rel 的计算公式相应为：

向前转移：rel=目的地址－（源地址+2）＝地址差－2

向后转移：rel＝（目的地址－（源地址+2））补＝FF－（源地址+2－目的地址）+1＝FE－地址差

此外，在等待中断或程序结束，常采用使“程序原地踏步”的方法：

HERE： SJMP HERE

或 HERE： SJMP $

指令的机器码是 80H，FE。以$代表 PC 当前的值。

3. 长跳转指令◆

LJMP addr16 ❍；PC ← addr16 ◆

执行该指令时, 将 16 位目标地址 addr16 装入 PC, 程序无条件转向指定的目标地址。转移的目标地址可以在 64 KB 程序存储器地址空间的任何地方, 不影响任何标志。 该指令是三字节指令，机器码为 02H，高 8 位地址，低 8 位地址。

4. 散转指令◆

JMP @A+DPTR◆；PC ← （A）+（DPTR）

该指令也叫变址寻址转移指令，是一条单字节指令。执行该指令是，把累加器 A 中的 8 位无符号数与数据指针 DPTR 中的 16 位数相加，结果作为下条指令的地址送入 PC。利用这条指令能实现程序的散转，即只要把 DPTR 的值固定，而给 A 赋以不同的值，从而可实现根据 A 的不同实现程序的多分支转移。

（二）条件转移指令

条件转移指令是当某种条件满足时, 程序转移执行；条件不满足时，程序仍按原来顺序继续执行。条件转移的条件可以是上一条指令或者更前一条指令的执行结果（常体现在标志位上），也可以是条件转移指令本身包含的某种运算结果。转移指令中的相对偏移量 rel 为 8 位带符号数，表示条件转移目标地址在以下条指令首地址为中心的 256B 范围内（-128～+127）。

汇编语言中一般靠 PSW 中的标志位或其他简单的方法来进行条件转移。

1. 累加器判零转移指令（2 条）

JZ rel ；若 (A)=0, 则 (PC) ← (PC)+2+rel

若 (A)≠0, 则 (PC) ← (PC)+2

JNZ rel ；若 (A)≠0, 则 (PC) ← (PC)+2+rel,

若 (A)=0, 则 (PC) ← (PC)+2

这组指令是依据累加器 A 的内容是否为 0 的条件转移指令。条件满足时转移（相当于一条相对转移指令），条件不满足时则顺序执行下面一条指令。转移的目标地址在以下一条指令的起始地址为中心的 256 个字节范围之内（－128～ +127）。当条件满足时，PC←(PC)+N+rel, 其中(PC)为该条件转移指令的第一个字节的地址, N 为该转移指令的字节数（长度），本转移指令 N=2。

例 1： 将外部数据 RAM 的一个数据块传送到内部数据 RAM，两者的首址分别为 DATA1 和 DATA2，遇到传送的数据为零时停止。

解： 外部 RAM 向内部 RAM 的数据传送一定要以累加器 A 作为过渡，利用判零条件转移正好可以判别是否要继续传送或者终止。完成数据传送的参考程序如下：

```
        MOV     R0, #DATA1  ; 外部数据块首址送 R0
        MOV     R1, #DATA2  ; 内部数据块首址送 R1
LOOP:  MOVX    A, @R0      ; 取外部 RAM 数据入 A
HERE:  JZ      HERE        ; 数据为零则终止传送
        MOV     @R1, A      ; 数据传送至内部 RAM 单元
        INC     R0          ; 修改地址指针, 指向下一数据地址
        INC     R1
        SJMP    LOOP        ; 循环取数
```

2. 比较转移指令（4 条）

CJNE　目的操作数, 源操作数, rel

这组指令是三字节指令。其功能是：比较前面两个操作数的大小，如果它们的值不相等则转移。转移地址的计算方法与上述两条指令相同。如果第一个操作数（无符号整数）小于第二个操作数，则进位标志 Cy 置“1”， 否则清“0”， 但不影响任何操作数的内容。

这组指令实现对两个规定的操作数进行比较，根据比较的结果来决定是否转移到目的地址。4 条比较转移指令如下：

```
CJNE   A, #data , rel     ; PC←（PC）+3，若(A)≠data，则 PC←（PC）+3+rel
CJNE   A, direct, rel     ; PC←（PC）+3，若(A)≠（direct），则 PC←（PC）+3+rel
CJNE   @Ri, #data, rel    ; PC←（PC）+3，若((Ri))≠data，则 PC←（PC）+3+rel
CJNER  Rn, #data, rel     ; PC←（PC）+3，若（Rn）≠data，则 PC←（PC）+3+rel
```

这 4 条指令的含义分别为：

第 1 条指令：累加器内容与立即数比较，不等则转移；

第 2 条指令：累加器内容与内部 RAM（包括特殊功能寄存器）内容比较，不等则转移；

第 3 条指令：内部 RAM 内容与立即数比较，不等则转移；

第 4 条指令：工作寄存器内容与立即数比较，不等则转移。

以上 4 条指令的差别仅在于操作数的寻址方式不同， 均完成以下操作：

若目的操作数=源操作数，则 (PC) ← (PC)+3 ；

若目的操作数>源操作数，则 (PC) ← (PC)+3+rel，CY=0；

若目的操作数<源操作数，则 (PC) ← (PC)+3+rel，CY=1；

偏移量 rel 的计算公式为：

向前转移： rel = FD－（源地址与目的地址差的绝对值）

向后转移： rel =（源地址与目的地址差的绝对值）-3

例如：当 P1 口输入为 3AH 时， 程序继续进行，否则等待，直至 P1 口出现 3AH。参考程序如下：

```
        MOV   A, #3AH          ; 立即数 3A 送 A
WAIT: CJNE   A, P1, WAIT       ; (P1)≠3AH, 则等待,
```

3. 减 1 条件转移指令

（循环转移指令）减 1 条件转移指令有如下两条：

```
DJNZ   direct, rel;       (direct)← (direct)－1
                          若(direct)=0, 则(PC) ← (PC)+3
                          否则, (PC) ← (PC)+3+rel
DJNZ   Rn, rel ;          (Rn) ← (Rn)－1
                          若(Rn)=0, 则(PC) ← (PC)+2
                          否则, (PC) ← (PC)+2+rel
```

这组指令是把减 1 功能和条件转移结合在一起的一组指令。程序每执行一次该指令， 就把第一操作数减 1，并且结果保存在第一操作数中， 然后判断操作数是否为零。 若不为零， 则转移到规定的地址单元，否则顺序执行。 转移的目标地址是在以 PC 当前值为中心的－128～+127 的范围内。 如果第

一操作数原为00H，则执行该组指令后，结果为FFH，但不影响任何状态标志。

这两条指令主要用于控制程序循环。如果预先把寄存器Rn或内部RAM单元direct赋值循环次数，则利用减1条件转移指令，以减1条件转移指令，以减1后是否为0作为转移条件，即可实现按次数控制循环。

例 2：延时子程序：

```
        MOV   R1, #0AH          ; 给R1赋循环初值
DELAY:  DJNZ  R1, DELAY         ; (R1) ← (R1)－1, 若(R1)≠0 则循环
```

由于DJNZ R1, DELAY为双字节双周期指令，当单片机主频为 12 MHz时，执行一次该指令需24个振荡周期约2μs。 因此，R1中置入循环次数为10时，执行该循环指令可产生20 μs的延时时间。

（三）子程序调用及返回指令

在主程序中，有时需要反复执行某段程序，通常把这段程序设计成子程序，用一条子程序调用指令，将程序转向子程序的入口地址。

通常把具有一定功能的公用程序段作为子程序而单独编写，当主程序需要引用这一子程序时，可利用调用指令对子程序进行调用。在子程序末尾安排一条返回指令，使子程序执行结束能返回到主程序。

1. 子程序调用指令

子程序调用有长调用指令LCALL和绝对调用指令ACALL两条， 它们分别表示如下：

（1）长调用指令

```
LCALL    addr16   ; (PC) ← (PC)+3
                  (SP) ← (SP)+1, ((SP)) ← (PC 7～0)
                  (SP) ← (SP)+1, ((SP)) ← (PC 15～8)
                  (PC) ← addr 15～0
```

这条指令无条件调用位于 16位地址addr16的子程序。执行该指令时，先将PC+3以获得下一条指令的首地址，并把它压入堆栈（先低字节后高字节）， SP内容加 2，然后将16位地址放入 PC中，转去执行以该地址为入口的程序。 LCALL指令可以调用64 KB范围内任何地方的子程序。指令执行后不影响任何标志。

例如：设（SP）=60H，标号START值为1000H，标号DIR值为4000H，执行指令

```
START:  LCALL   DIR
```

指令执行结果为：（SP）=62H，（61H）=03H，（62H）=10H，（PC）=4000H。

（2）绝对调用指令

```
ACALL    addr11  ; (PC) ← (PC)+2
                 (SP) ← (SP)+1, ((SP)) ← (PC 7～0)
                 (SP) ← (SP)+1, ((SP)) ← (PC 15～8)
                 (PC) 10～0 ← addr11
```

这是一条 2 KB范围内的子程序调用指令。执行该指令时，先将PC+2以获得下一条指令的地址，然后将 16位地址压入堆栈（PCL内容先进栈， PCH内容后进栈），SP内容加 2，最后把 PC的高5位PC15～PC11与指令中提供的 11位地址addr11相连接（PC15～PC11， A 10～A 0），形成子程序的入口地址送入PC，使程序转向子程序执行。所用的子程序的入口地址必须与 ACALL下面一条指令的第一个字节在同一个2 KB区域的存储器区内。

例如：设（SP）=60H，标号NBA值为0123H，子程序START位于0345H，执行指令

```
NBA:  LCALL   START
```

指令执行结果为：（SP）=62H，（61H）=25H，（62H）=01H，（PC）=0345H。

LCALL和ACALL指令类似于转移指令LJMP和AJMP，不同之处在于它们在转移前要把执行完该指令的PC内容自动压入堆栈后，才将addr16（或addr11）送往PC，即把子程序的入口地址装入PC。

2. 返回指令

返回指令共有两条： 一条是对应两条调用指令的子程序返回指令RET； 另一条是对应从中断服务程序的返回指令RETI。

（1）子程序返回指令

RET　　；（PC15～8）← ((SP))，(SP) ← (SP)－1

（PC7～0）← ((SP))，(SP) ← (SP)－1

这条指令的功能是：恢复断点，将调用子程序时压入堆栈的下一条指令的首地址取出送入 PC，使程序返回主程序继续执行。

例如：设（SP）=62H，（62H）=10H，（61H）=30H，执行指令

RET

指令执行结果为（SP）=60H，（PC）=1030H。

（2）中断返回指令

RETI　　；（PC15～8）← ((SP))，(SP) ← (SP)－1

（PC7～0）← ((SP))，(SP) ← (SP)－1

这条指令的功能与 RET 指令相似，除具有上述子程序返回指令所具有的全部功能外，还要清除中断响应时被置位的优先级状态、开放较低一中断和恢复中断逻辑功能。

3. 空操作指令

NOP　　；(PC) ← (PC)+1

在执行这条指令时，CPU 不作任何操作，仅消耗一个机器周期的时间。NOP 指令常用于子程序的等待或时间的延迟。

3.3.5 位操作类指令

位操作又称为布尔变量操作，它是以位(bit)作为单位来进行运算和操作的。 MCS－51 系列单片机内设置了一个位处理器（布尔处理机），它有自己的累加器（借用进位标志 CY）， 自己的存储器（即位寻址区中的各位），也有完成位操作的运算器等，如表 3.7 所示。

表 3.7　　位操作指令

指令助记符	功能简述	字节数	振荡器周期数
MOV C，bit	Cy ←(bit)	2	12
MOV bit，C	bit ←Cy	2	24
CLR C	Cy ←0	1	12
CLR bit	bit ←0	2	12
CPL C	Cy ←(Cy)	1	12
CPL bit	bit ←(bit)	2	12
SETB C	Cy ←1	1	12
SETB bit	bit ←1	2	12
ANL C，bit	Cy ←(Cy)∧(bit)	2	24
ANL C，/bit	Cy ←(Cy)∧(bit)	2	24
ORL C，bit	Cy ←(Cy)∨(bit)	2	24
ORL C，/bit	Cy ←(Cy)∨(bit)	2	24
JC rel	若(Cy)=1，则转移，PC←(PC)+2+rel	2	24
JNC rel	若(Cy)=0，则转移，PC←(PC)+2+rel	3	24
JB bit，rel	若(bit)=1，则转移，PC←(PC)+3+rel	3	24
JNB bit，rel	若(bit)=0，则转移，PC←(PC)+3+rel, 并 bit=0	3	24

（一） 位传送指令

MOV C, bit◆　　；C ←（bit）

MOV bit, C◆　　；bit ← C

这组指令的功能是：把源操作数指出的布尔变量送到目的操作数指定的位地址单元中。其中一个操作数必须为进位标志 Cy，另一个操作数可以是任何可直接寻址位。

由于没有两个可寻址位之间的传送指令，如要完成这种传送，必须使用 Cy。例如，将 50H 位的内容传送 20H 位。

```
MOV   10H,C      ；暂存 Cy 内容
MOV   C，50H     ；50H 位送 Cy
MOV   20H，C     ；Cy 送 20H
MOC   C,10H      ；恢复 Cy 内容
```

（二） 位赋值指令

```
CLR     bit     ；(bit) ←0
CLR      C      ；(CY) ←0
SETB    bit     ；(bit) ←1
SETB     C      ；(CY) ←1
```

这组指令对操作数所作出的位进行清“0”，取反，置“1”的操作，不影响其它标志。

（三）位逻辑运算指令

1. 位变量逻辑与指令

```
ANL     C, bit     ; (CY) ← (CY)∧(bit  )
ANL     C, / bit   ; (CY) ← (CY)∧( /bit )
```

这组指令的功能是： 如果源位的布尔值是逻辑 0， 则将进位标志清“0”； 否则，进位标志保持不变，不影响其它标志。bit 前的斜杠表示对(bit)取反， 直接寻址位取反后用作源操作数，但不改变直接寻址位原来的值。

例如指令： ANL C, /ACC.0 执行前 ACC.0 为 0, C 为 1，则指令执行后 C 为 1， 而 ACC.0 仍为 0。

2. 位变量逻辑或指令

```
ORL     C, bit◆    ; (CY) ← (CY)∨( bit )
ORL     C, /bit◆   ; (CY) ← (CY)∨(/ bit )
```

这组指令的功能是： 如果原位的布尔值是逻辑 1， 则将进位标志置“1”； 否则， 进位标志保持不变，不影响其它标志。

3. 位变量逻辑非指令

```
CPL     bit        ; (bit) ← ( bit)
CPL      C         ; (Cy) ← (Cy)
```

这组指令的功能是：将源位的布尔值取反后，再回送给原位。

例如：利用位逻辑指令, 模拟图 3.6 所示硬件逻辑电路 功能。 参考子程序如下:

```
PR2:  MOV  C, P1.1        ; (CY) ← (P1.1)
      ORL   C, P1.2       ; (CY) ← (P1.1)∨(P1.2) = A
      ANL   C, P1.0       ; (CY) ← (P1.0)∧A
      CPL   C             ; (CY) ←
      MOV   F0, C         ; F0 内暂存 B
      MOV   C, P1.3       ; (CY) ← (P1.3)
      ANL   C, /P1.4      ; (CY) ← (P1.3)∧
      ORL   C , F0        ; (CY) ← B∨D
      MOV   P1.5, C       ; 运算结果送入 P1.5
      RET
```

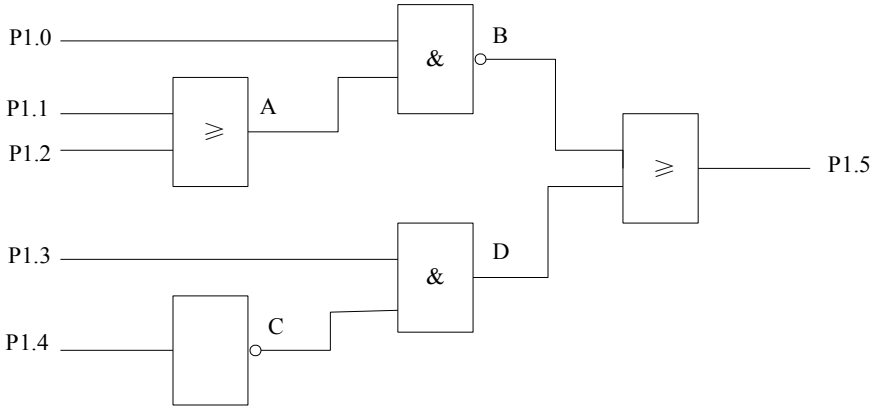


图 3.6　硬件逻辑电路

（四）位转移指令

位条件转移指令是以进位标志 CY 或者位地址 bit 的内容作为是否转移的条件，共有 5 条指令。分别检测指定位是 1 还是 0，若条件符合，则 CPU 转向制定的目标地址去执行程序，否则，顺序执行下条指令。

(1) 以 CY 内容为条件的双字节双周期转换指令

JC　rel　；若(CY)=1, 则(PC) ← (PC)+2+rel 转移
　　否则, (PC) ← (PC)+2 顺序执行

JNC　rel　；若(CY)=0, 则(PC) ← (PC)+2+rel 转移
　　否则, (PC) ← (PC)+2 顺序执行

这两条指令常和比较条件转移指令 CJNE 一起使用，先由 CJNE 指令判别两个操作数是否相等，若相等就顺序执行；若不相等则依据两个操作数的大小置位或清零 CY(参见图 3.8)，再由 JC 或 JNC 指令根据 CY 的值决定如何进一步分支, 从而形成三分支的控制模式，如图 3. 7 所示。

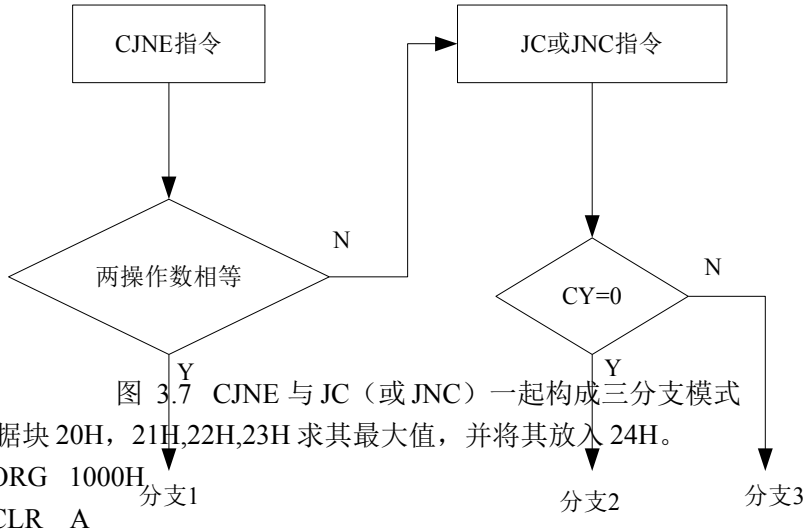
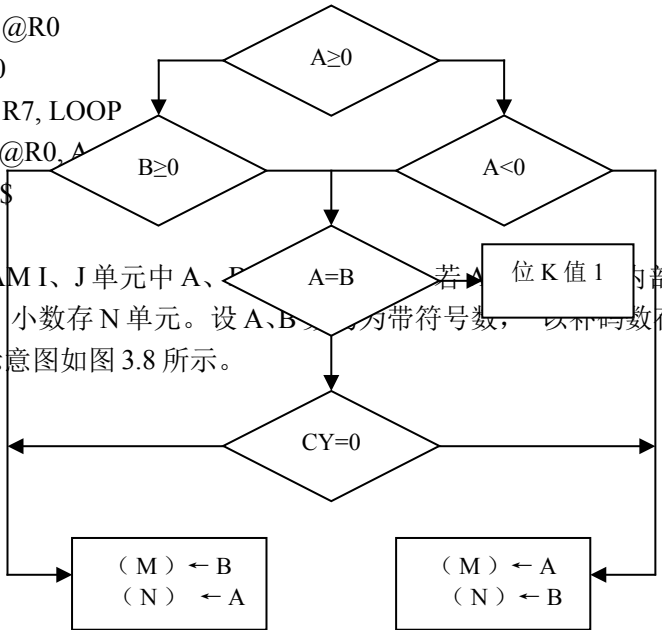


图 3.7　CJNE 与 JC（或 JNC）一起构成三分支模式

例 2：有一数据块 20H，21H,22H,23H 求其最大值，并将其放入 24H。

```
        ORG   1000H
        CLR   A
        MOV   R0, #20H
        MOV   R7, #04H
LOOP:   CLR   C
        SUBB  A, @R0
        JNC   NEXT
        MOV   A, @R0
        SJMP  NEXT1
NEXT:   ADD   A, @R0
NEXT1:  INC   R0
        DJNZ  R7, LOOP
        MOV   @R0, A
        SJMP  $
        END
```



例 3：　比较内部 RAM I、J 单元中 A、B 两数大小，若 A=B，则内部 RAM 的位 K 置 1；若 A≠B，则大数存 M 单元，小数存 N 单元。设 A、B 为带符号数，以补码数存入 I、J 中，该带符号数比较子程序的比较过程示意图如图 3.8 所示。

图 3.8 带符号数比较过程示意图

参考子程序如下：

```
        MOV    A, I              ; A 数送累加器 A
        ANL    A, ＃80H          ; 判 A 数的正负
        JNZ    NEG               ; A<0 则转至 NEG
        MOV    A, J              ; B 数送累加器 A
        ANL    A, ＃80H          ; 判 B 数的正负
        JNZ    BIG1              ; A≥0, B＜0, 转 BIG1
        SJMP   COMP              ; A≥0, B≥0, 转 COMP
NEG:    MOV    A, J              ; B 数送累加器 A
        ANL    A, ＃80H          ; 判 B 数的正负
        JZ     SMALL             ; A＜0, B≥0, 转 SMALL
COMP:   MOV    A, I              ; A 数送累加器 A
        CJNE   A, J, BIG         ; A≠B 则转 BIG
        SETB   K                 ; A=B, 位 K 置 1
        RET
BIG:    JC     SMALL             ; A<B 转 SMALL
BIG1:   MOV    M, I              ; 大数 A 存入 M 单元
        MOV    N, J              ; 小数 B 存入 N 单元
        RET
SMALL:  MOV    M, J              ; 大数 B 存入 M 单元
        MOV    N, I              ; 小数 A 存入 N 单元
        RET
```

（2） 以位地址内容为条件的三字节双周期转移指令

```
JB    bit, re l     ; 若(bit)=1, 则(PC) ← (PC)+3+rel 转移
                            否则, (PC) = (PC)+3 顺序执行
JNB   bit, rel      ; 若(bit)=0, 则(PC) ← (PC)+3+rel 转移
                            否则, (PC) ← (PC)+3 顺序执行
JBC   bit, rel      ; 若(bit)=1, 则(PC) ← (PC)+3+rel, (bit)←0
                            否则, (PC) ← (PC)+3 顺序执行
```

上述指令测试直接寻址位，若位变量为 1（第 1、3 条指令）或位变量为 0（第 2 条指令）， 则程序转向目的地址去执行， 否则顺序执行下条指令。 该类指令测试位变量时，不影响任何标志。 前两条指令执行后也不影响原位变量值，而第 3 条指令虽和第 1 条指令的转移功能相同，但无论测试位变量原为何值，检测后即对该位变量清零。该指令直接寻址位若为某端口的某位时，具有读—修改—写功能。

应该注意的是，指令中地址的表达形式有以下几种：

(1) 直接位地址表示方式：如 0A8H；

(2) 点操作符表示（说明是什么寄存器的什么位）方式： 如 PSW.5， 说明是 PSW 的第五位；

(3) 位名称表示方法： 如 EX0；

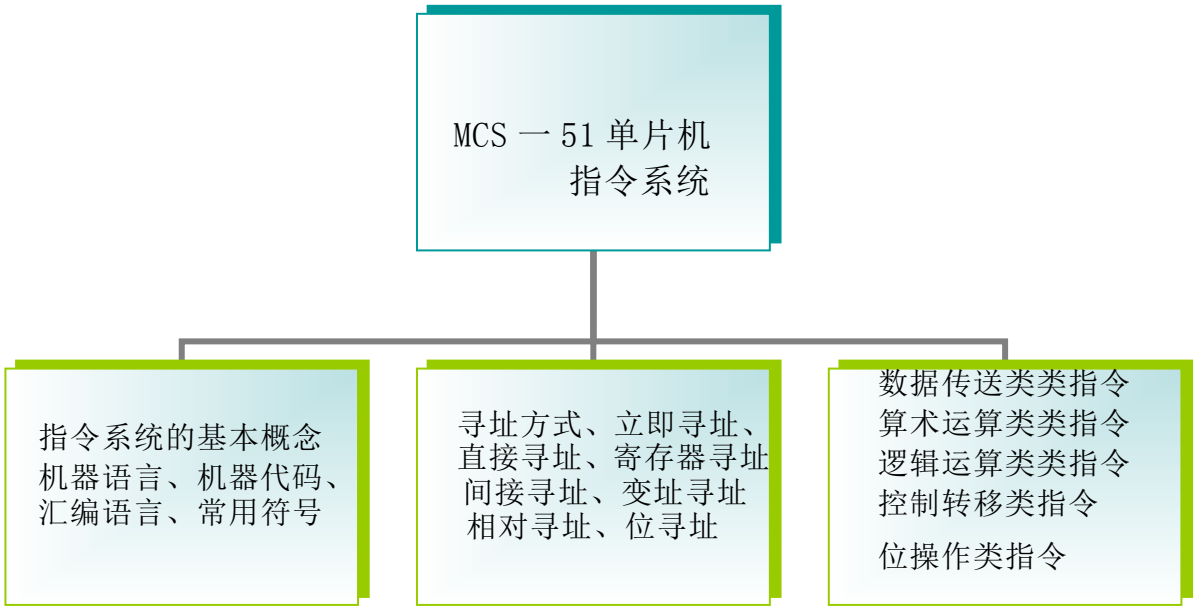
(4) 用户定义名表示方式： 如用伪指令 BIT 定义：

WBZD0 BIT EX0

经定以后，允许指令中使用 WBZD0 代替 EX0。

以上介绍了 MCS－51 系列单片机的指令系统。有关 111 条指令助记符、操作数、机器代码以及字节数和指令周期一览表详见附录 3。

重点串联



技能训练:

【课时建议】 4 课时

实训 3.1 指令助记符转换为机器码

1. 实训目的

（1）更进一步理解和熟悉单片机汇编指令系统。

（2）了解并掌握手工汇编的过程。

（3）了解并掌握机器汇编的过程。

（4）明确汇编语言程序的执行过程。

2. 实训内容

将程序翻译成机器码，可以通过查阅单片机指令系统表获得机器码（即手工汇编），也可以通过 WAVE6000 软件经过编译得到机器码（即机器汇编）。

3. 实训要求

（1）了解机器码形成的过程。

（2）会编写简单的汇编语言程序。

（3）掌握源程序调试的方法。

（4）掌握汇编语言程序编译软件 WVAE6000 的安装和使用。

实训 3.2　　将汇编后的指令下载到单片机开发系统中

1. 实训目的

（1）了解单片机开发系统的基本组成及功能。

（2）通过简单的应用了解单片机开发系统的使用方法。

（3）了解编程器的使用及编程器软件的安装

（4）明确汇编语言程序的执行过程。

2. 实训内容

（1）在仿真软件 proteus7.5 中绘制好单片机控制的彩灯电路图。

（2）在编译软件 WAVE6000 中编写好单片机控制的彩灯源程序。

（3）将 WAVE6000 中编译形成的.hex 文件加载到仿真软件的电路图中。

（4）运行单片机控制的彩灯系统。

3. 实训要求

（1）了解输入目标代码的方法。

（2）掌握输入汇编语言程序的方法。

（3）掌握加载目标代码的方法。

（4）掌握调试程序的方法。

（5）掌握检查寄存器和存储器中内容的方法。

（6）掌握编程器的使用。

实训 3.3　数码管显示器的设计

1. 实训目的

（1）了解七段数码管的引脚结构及工作原理。

（2）掌握七段数码管与单片机的接口电路及相应的接口编程。

（3）了解集成块 74LS07 的使用。

（4）掌握共阴极和共阳极七段数码管对十六进制数 0-9、A-F 的编码

2. 实训内容

（1）用共阴极七段数码显示数字的变化。

（2）设计一个单片机控制的篮球比赛记分牌系统，并编写相应的程序。

3. 实训要求

（1）查看所用的单片机开发系统的按键、七段数码管。

（2）给出实训内容的硬件电路和程序清单。

（3）对所设计的硬件和软件进行实际调试

（4）观察、分析并记录实训结果

（5）撰写实训报告

思考与练习

1、89C51 系列单片机的指令系统有何特点？

2、89C51 单片机有哪几种寻址方式？各寻址方式所对应的寄存器或存储器空间如何？

3、访问特殊功能寄存器 SFR 可以采用哪些寻址方式？

4、访问内部 RAM 单元可以采用哪些寻址方式？

5、访问外部 RAM 单元可以采用哪些寻址方式？

6、访问外部程序存储器可以采用哪些寻址方式？

7、指出在下列各条指令中，30H 分别代表什么含义？

```
MOV   A, #30H
MOV   A, 30H
MOV   30H, 30H
MOV   30H, 28H
MOV   C, 30H
```

8、试根据指令编码表写出下列指令的机器码。

（1）MOV A，#88H

（2）MOV R3，50H

（3）MOV P1.1，#55H

（4）ADD   A，@R1

（5）SETB 12H

9、完成某种操作可以采用几条指令构成的指令序列实现，试写出完成以下每种操作的指令序列。

（1）将R0的内容传送到R1；

（2）内部RAM单元60H的内容传送到寄存器R2；

（3）外部RAM单元1000H的内容传送到内部RAM单元60H；

（4）外部RAM单元1000H的内容传送到寄存器R2；

（5）外部RAM单元1000H的内容传送到外部RAM单元2000H。

10、若（R1）=30H，（A）=40H，（30H）=60H，（40H）=08H。试分析执行下列程序段后上述各单元内容的变化。

```
MOV  A，@R1
MOV  @R1，40H
MOV  40H，A
MOV   R1，#7FH
```

11、若（A）=E8H，（R0）=40H，（R1）=20H，（R4）=3AH，（40H）=2CH，（20）=0FH，试写出下列各指令独立执行后有关寄存器和存储单元的内容？若该指令影响标志位，试指出CY、AC、和OV的值。

（1）MOV A，@R0

（2）ANL 40H，#0FH

（3）ADD A，R4

（4）SWAP A

（5）DEC @R1

（6）XCHD   A，@R1

12、若（50H）=40H，试写出执行以下程序段后累加器A、寄存器R0及内部RAM的40H、41H、42H单元中的内容各为多少？

```
MOV  A，50H
MOV  R0，A
MOV  A，#00H
MOV  @R0，A
MOV  A，3BH
MOV  41H，A
MOV  42H，41H
```

13、试用位操作指令实现下列逻辑操作。要求不得改变未涉及的位的内容。

（1）使ACC.0置位；

（2）清除累加器高4位；

（3）清除ACC.3，ACC.4，ACC.5，ACC.6。

14、 试写出达到下列要求的程序

（1）将外部RAM 1000H单元中的低4位清0，其余位不变，结果存回原处。

（2）将内部RAM 50H 单元中的高3位置1，其余位不变，结果存回原处。

（3）将内部RAM 20H 单元中的高4位置1，低4位清0，结果存回原处。

（4）将DPTR 的中间8位取反，其余位不变，结果存回原处。

15、 SJMP指令和LJMP指令都是双字节转移指令，它们有什么区别？各自的转移范围是多少？能否用AJMP代替SJMP？为什么？

16、若（SP）=35H，（PC）=2345H，标号LOOP所在的地址为3456H。执行长调用指令“LCALL LOOP”后，堆栈指针和堆栈的内容发生什么变化？PC的值为多少？若将上述指令改为“ACALL LOOP”是否可以？为什么？

17、试编写程序，将内部RAM的20H、21H、22H三个连续单元的内容依次存入2FH、2EH和2DH单元。

18、试编程实现：查找内部RAM 20H—50H单元中出现00H的次数，并将查找结果存入R1中。

19、若（CY）=1，（P1）=10100011B，（P3）=01101100B。试指出执行下列程序段后，CY、P1口及P3口内容的变化情况。

```
MOV  P1.3，C
MOV  P1.4，C
MOV  C，P1.6
MOV  P3.6，C
MOV  C，P1.0
MOV  P3.4，C
```